DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Intro to DCL Programming

David J. Dachtera
djesys@fsi.net

DJE Systems
http://www.djesys.com/

This presentation provides an introduction to programming in Digital Command Language - DCL.

DCL includes features allowing for conditional statements, logical control, file I/O string and numeric manipulation, and more.

In this presentation we'll introduce some of the more basic features of DCL as a programming language.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Agenda

- Basic DCL Concepts
  - Commands
  - Verbs
  - Symbols
- IF-THEN
- IF-THEN-ENDIF
- IF-THEN-ELSE-ENDIF
- Labels, GOTO

In this presentation, we'll go over some DCL basics: commands, verbs and symbols.

We'll look at conditional statements and conditional statement blocks.

We'll look at logical control and how to pass control from one section of code to another.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Agenda, Cont'd

- GOSUB-RETURN
- Common Lexical Functions
  - F$CVTIME
  - F$GETDVI
  - F$GETJPI
  - F$GETQUI
  - F$GETSYI
- PARAMETERS
- Logical Names

DECUS Symposium - Fall 1999 San Diego Slide 3

Then, we'll look at some of the more modular structures, including internal subroutines, using the GOSUB and RETURN statements.

We'll look at some functions built into DCL that allow you to manipulate dates and times, get information about devices, processes, batch and print queues and even from the system itself.

We'll look at passing parameters to DCL procedures, getting information from logical names and symbols, ...

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Agenda, Cont'd

Batch jobs that reSUBMIT themselves
Daily Jobs
Weekly Jobs
Question & Answer

Finally, we'll use what we've learned to build batch jobs which resubmit themselves, and change their behavior based on the day of the week, month or year.

We'll also have a question and answer session to help everyone understand everything they need to help make their job easier.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

DCL - Programming?

DCL as a programming language?

Never thought of DCL as a programming language?

Well, that's even what Digital once said - but not any more!

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

DCL Command Elements

$ verb parameter_1 parameter_2

DCL Commands consist of a verb and one or more parameters.

DECUS Symposium - Fall 1999 San Diego Slide 6

To begin understanding DCL, first let's review a few basic concepts.

DCL Commands consist of a verb and one or more parameters or operands.

Some commands can be as simple as:

$ SET VERIFY

or

$ LOGOUT

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

DCL Verbs

Internal commands

ASSIGN, CALL, DEFINE, GOSUB, GOTO, IF, RETURN, SET, STOP, others…

External commands

APPEND, BACKUP, COPY, DELETE, PRINT, RENAME, SET, SUBMIT, others...

Commands in DCL are either internal to DCL or are executed by programs which are external to DCL.

Here we see some examples of both internal and external commands.

Notice that SET, STOP and other commands can be either internal or external depending upon the keyword after the verb.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

DCL Verbs, Cont'd

"Foreign" Commands

```
$ symbol = value
```

Examples:

```
$ DIR :== DIRECTORY/SIZE=ALL/DATE
$ ZIP :== $ZIP/VMS
```

DECUS Symposium - Fall 1999 San Diego Slide 8

Commands can be added or customized using symbols or "foreign commands".

In the slide, the DIR symbol redefines the behavior of the DIRECTORY command, while the ZIP symbol provides a means to invoke the ZIP program in such a manner that it can accept parameters and qualifiers from the command line.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Command Qualifiers

```
$ command/qualifier

$ command/qualifier=value
$ command/qualifier=(value,value)

$ command/qualifier=keyword=value
$ command/qualifier=-
    (keyword=value,keyword=(value,value))
```

DECUS Symposium - Fall 1999 San Diego Slide 9

Command qualifiers specify additional information or alternate behaviors of commands.

Some qualifiers accept a value or a list of values. When specifying a single value, the parentheses can be left out.

Some qualifiers accept a keyword or a list of keywords. Each keyword may accept a value or a list of values.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Non-positional Qualifiers

Apply to the entire command, no matter where they appear.

```
$ command param1/qual param2
```

Example:

```
$ COPY A.DAT A.NEW/LOG
$ DELETE/LOG C.TMP;
```

DECUS Symposium - Fall 1999 San Diego Slide 10

Some qualifiers have the same effect no matter where they appear on the command line. These are called non-positional qualifiers.

The slide shows some examples. The /LOG qualifier is usually non-positional.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Positional Qualifiers

Apply only to the object they qualify.

```
$ command param1/qual=value1 -
    param2/qual=value2
```

Examples:

```
$ PRINT/COPIES=2 RPT1.LIS, RPT2.LIS
$ PRINT RPT1.LIS/COPIES=1,-
    RPT2.LIS/COPIES=3
```


Some qualifiers can appear more than once in a command. These are called positional qualifiers. They qualify (or modify) the command element to which they are immediately adjacent.

An example of this is the /COPIES qualifier of the PRINT command. When applied to the PRINT command, it is global to all the files in the print job. When applied to single file specifications in a PRINT job, it modifies only those files which match that file specification.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Common Qualifiers

Many commands support a set of common qualifiers:

/BACKUP /BEFORE /CREATED
/EXCLUDE /EXPIRED /INCLUDE
/MODIFIED /OUTPUT /PAGE /SINCE

See the on-line HELP for specifics.

DECUS Symposium - Fall 1999 San Diego Slide 12

The VMS run time library UTIL$SHR provides support for a set of common qualifiers that have been made available in many of the more common commands.

You can find these in the HELP for the DIRECTORY command, SEARCH, PRINT and others.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

DCL Statement Elements

$ vbl = value

DCL statements are typically assignments where a variable receives a value.

Elements of a DCL statement (as opposed to a command) look very much like other programming languages.

Here we have an example of an assignment statement. A variable receives a value. The value can be a literal expression, the name of another symbol, the result of a function, the result of an arithmetic or string operation, etc.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Assignment Statements

$ vbl = F$lexical_function(params)

Examples:

$ FSP = F$SEARCH("*.TXT")
$ DFLT = F$ENVIRONMENT ("DEFAULT")
$ NODE = F$GETSYI("NODENAME")

DECUS Symposium - Fall 1999 San Diego Slide 14

Here we see a variable which receives the value returned by a built-in ("lexical") function. The built-in function is part of the DCL lexicon.

The slide also shows some examples.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Assignment Statements

$ vbl = string_expression

Examples:

```
$ A = “String 1 “ + “String 2”
$ B = A - “String “ - “String “
$ C = ‘A’
```

Maximum string length = 255 bytes

DECUS Symposium - Fall 1999 San Diego Slide 15

Here are some examples of string operations.

The first operation is a string concatenation.

The second operation is string reduction.

The third operation is a symbol substitution.

What’s happening in statement three?

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Assignment Statements

$ vbl = numeric_expression

Examples:

```
$ A = 1
$ B = A +1
$ C = B + A + %X7F25
$ D = %O3776
```


Here, we see some examples of numeric assignments.

We have an assignment using a literal and other assignments using numeric additions.

Note the use of hexadecimal notation in the third example and octal notation in the fourth.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Assignment Statements

```
$ vbl[start_bit,bit_count]=numeric_exp
```

Examples:

```
$ ESC[0,8]=%X1B
$ CR[0,8]=13
$ LF[0,8]=10
$ FF[0,8]=12
$ CRLF[0,8]=13
$ CRLF[8,8]=10
```

DECUS Symposium - Fall 1999 San Diego Slide 17

These are examples of assigning values to bits within a string. The result is always a string. This is useful for constructing escape sequences and binary values.

In the fifth and sixth examples, the result is a two byte string containing a carriage-return and a line-feed (in that order).

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Assignment Statements

$ vbl = boolean_expression

Examples:

```
$ MANIA = (“TRUE” .EQS. “FALSE”)
$ TRUE = (1 .EQ. 1)
$ FALSE = (1 .EQ. 0)
$ YES = 1
$ NO = 0
```

DECUS Symposium - Fall 1999 San Diego Slide 18

Here we have examples of assignment of a “truth value” or a boolean value.

The last two examples are ordinary numeric literal assignments. They illustrate the defaults for “true” (“yes”) and “false” (“no”).

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Assignment Statements

Local Assignment:

```
$ vbl = value
```

Global Assignment:

```
$ vbl == value
```


Symbols can be either local to the current procedure level (“depth”) and all levels deeper, or global to all procedure levels (“depths”).

Local symbols are available to the current procedure and any that it invokes.

Global symbols are available to the current procedure and any that it invokes, as well as the procedure(s) which invoked the current procedure.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Assignment Statements

Quoted String:

$ vbl = “quoted string”

Case is preserved.

Examples:

$ PROMPT = “Press RETURN to continue “

$ INVRSP = “% Invalid response!”

When a quoted string is assigned to a symbol, the case and contents of the string are preserved intact.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Assignment Statements

Unquoted string:

- $ vbl := unquoted string
  - Case is **NOT** preserved, becomes uppercase. Leading/trailing spaces are trimmed off.

Examples:

- $ SAY := Write Sys$Output
- $ SYSMAN :== $SYSMAN ! Comment

In the case of an unquoted string (uses the “colon-equal[-equal]” sequence), all text becomes upper case, and leading and trailing spaces and TABs are trimmed off. If the unquoted string contains an embedded quoted string, the case and content of the quoted portion of the string will be preserved.

Comment delimiters are observed as usual. The comment is not considered part of the unquoted string.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Foreign Commands

$ vbl := $filespec[param[param[…]]]

“filespec” defaults to SYS$SYSTEM:.EXE

Maximum string length is 510 bytes.

A “foreign command” is special case where a symbol can be interpreted by DCL as verb. The value of the symbol can include qualifiers and/or parameters in addition to the file specification of the executable file.

If necessary, foreign commands can be defined using quoted strings if, for example, the case of an argument or embedded spaces within an argument needs to be preserved.

Using symbol substitution, strings of up to 1024 bytes can be constructed.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Conditional Expressions

$ IF condition THEN statement

Variations:
$ IF condition THEN $ statement
$ IF condition THEN -
$ statement

DECUS Symposium - Fall 1999 San Diego Slide 23

Conditional expressions provide logical control based on conditions you specify.

In this form, the IF-THEN structure can be stated on a single line or it can be continued across two or more lines.

In either case, the “$” after THEN is optional.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Conditional Expressions

```
$ IF condition
$ THEN
$     statement(s)
$ ENDIF
```

DECUS Symposium - Fall 1999 San Diego Slide 24

Another variation is the IF-THEN[-ELSE]-ENDIF structure.

This variant allows multiple statements (or no statements) to be included in the THEN or ELSE clause.

Although it is not required, it is recommended that the THEN or ELSE statement appear on a line by itself.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Conditional Expressions

```
$ IF condition
$ THEN
$     IF condition
$     THEN
$           statement(s)
$     ENDIF
$ ENDIF
```


The IF-THEN[-ELSE]-ENDIF structure allows IF-THEN[-ELSE]-ENDIF structures to be nested.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Conditional Expressions

```
$ IF condition
$ THEN
$     IF condition
$     THEN
$          statement(s)
$     ENDIF
$     statement(s)
$ ENDIF
```

DECUS Symposium - Fall 1999 San Diego Slide 26

Other statements can be included either before or after a nested IF-THEN[-ELSE]-ENDIF structure.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Conditional Expressions

```
$ IF condition
$ THEN
$     statement(s)
$     IF condition
$     THEN
$          statement(s)
$     ENDIF
$ ENDIF
```


If one or more statements are included before a nested IF-THEN[-ELSE]-ENDIF structure, it is recommended that the preceding THEN or ELSE statement appear on a line by itself. In some older versions of VMS, this is a requirement.

For current and future versions of VMS, it is recommended that this guideline be observed to prevent your procedures from “breaking” due to a VMS upgrade, or due to being used on an older VMS version.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Conditional Expressions

```
$ IF condition
$ THEN statement(s)
$       IF condition
$       THEN
$               statement(s)
$       ENDIF
$ ENDIF
```

This may not work in pre-V6 VMS!

Here’s an example of some code that might not work in some older versions of OpenVMS.

Notice that the THEN clause includes a DCL statement, instead of being on a line by itself.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Conditional Expressions

```
$ IF condition
$ THEN
$      statement(s)
$ ELSE
$      statement(s)
$ ENDIF
```


Here is an example of an IF-THEN-ELSE-ENDIF block.

As we discussed earlier, the THEN and ELSE statements are recommended to be on lines by themselves.

Either the THEN or ELSE portions may contain nested IFs of any kind.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Labels, GOTO

```
$ GOTO label_1
   .
   .
   .
$label_1:
```


The GOTO statement provides for logical control within your procedures.

Combined with IF, GOTO provides for powerful logical control within your procedures.

Labels are defined by including the label name on a line followed immediately by a colon.

Any statement can follow a label on a line; however, this is recommended only for he SUBROUTINE statement. Otherwise, place the label on a line by itself for readability.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

GOSUB, RETURN

```
$ GOSUB label_1
   .
   .
   .
$label_1:
$ statement(s)
$ RETURN
```

DECUS Symposium - Fall 1999 San Diego Slide 31

The GOSUB and RETURN statements let you create internal subroutines.

All symbols local to the current procedure level are available, as are all global symbols.

Combined with IF, GOSUB and RETURN provide for powerful logical control within your procedures.

Labels are defined by including the label name on a line followed immediately by a colon.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

SUBROUTINE - ENDSUB...

```
$ CALL label_1[ param[ param[ …]]
   .
   .
   .
$label_1: SUBROUTINE
$ statement(s)
$ END SUBROUTINE
```


Another form of subroutine is enclosed within the SUBROUTINE and ENDSUBROUTINE statements. This form of subroutine is similar to invoking an external procedure.

Use the CALL statement to invoke this form of internal subroutine. Optionally, parameters to be passed to the subroutine can be included on the CALL statement.

We'll discuss this further in the Intermediate DCL Programming Session.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Common Lexical Functions

$ vbl = F$CVTIME(string[, keyword[, keyword]])

"string" = Absolute time expression

"keyword" = (1st instance) is one of "ABSOLUTE", "COMPARISION", "DELTA"

"keyword" = (2nd instance) is one of "DATE", "DATETIME", "DAY", "MONTH", "YEAR", "HOUR", "MINUTE", "SECOND", "HUNDREDTH", "WEEKDAY"

DECUS Symposium - Fall 1999 San Diego Slide 33

The DCL lexicon includes a number of built-in functions. Let's look at a few of the more useful ones…

F$CVTIME() can be used to develop routines to get and compare elements of the system date/time.

This can be useful for procedures that need to change their behavior based on the day, date and/or time of day.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Common Lexical Functions

F$CVTIME(), Continued…

Defaults:

```
$ vbl = F$CVTIME(string, -
        ”COMPARISON”, -
        ”DATETIME” )
```


This slide shows the default behavior of F$CVTIME() if no arguments are provided.

The default value for “string” is the current date and time.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Common Lexical Functions

F$CVTIME(), Continued…

Date Formats:

- Comparison
  - YYYY-MM-DD HH:MM:SS.CC
- Absolute
  - DD-MMM-YYYY HH:MM:SS.CC
- Delta
  - +/-DDDDD HH:MM:SS.CC

DECUS Symposium - Fall 1999 San Diego Slide 35

This slide illustrates the date/time formats used and returned by F$CVTIME().

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Common Lexical Functions

$ vbl = F$GETDVI(dev_name, keyword)

“dev_name” is a valid device name

“keyword” is a quoted string

Examples:

```
$ FBLK = F$GETDVI( “DUA0”,”FREEBLOCKS”)
$ MNTD = F$GETDVI( “DKA500”,”MNT”)
$ DVNM := DUA0:
$ VLNM := VOLNAM
$ VNAM = F$GETDVI( DVNM, VLNM )
```


The F$GETDVI() function is useful for retrieving information about system devices and disk/tape volumes.

The examples show some of the information that can be returned by F$GETDVI(). Notice that either literal strings or symbols can be used as arguments to DCL lexical functions.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Common Lexical Functions

```
$ vbl = F$QETQUI( -
        function,-
        item,-
        value,-
        keyword(s))
```

See the on-line help for descriptions.

DECUS Symposium - Fall 1999 San Diego Slide 37

F$GETQUI() is a useful, if rather complex lexical function.

We won’t go into great detail about it in this session. Later in this session, we will show how a batch job can use F$GETQUI() to get information about itself.

See the on-line HELP and the DCL Dictionary for further information about F$GETQUI().

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Common Lexical Functions

$ VBL = F$GETJPI(pid, keyword)

Examples:

$ USN = F$GETJPI(0, “USERNAME”)

$ MOD = F$GETJPI(0, “MODE”)

F$GETJPI() can be used to get information about your own process or about any other process to which you have access. Normal rules of OpenVMS privilege apply.

For information about the current process, specify the PID argument as a zero(0) as shown in the examples, or as a null string.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Common Lexical Functions

```
$ vbl = F$GETSYI( item[, nodename] )
```

Examples:

```
$ NODE = F$GETSYI( “NODENAME” )
$ FGP = F$GETSYI( “FREE_GBLPAGES” )
$ FGS = F$GETSYI( “FREE_GBLSECTS” )
```


F$GETSYI() can be used to retrieve information about the running system. In some cases, it can also be used to get information about other members of an OpenVMS cluster.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Parameters

$ @procedure_name p1 p2 p3 … p8

Notes:

- Only eight(8) parameters are passed from the command line, P1 through P8
- Parameters with embedded spaces must be quoted strings.
- Parameters are separated by a space.

DECUS Symposium - Fall 1999 San Diego Slide 40

This slide shows how to pass parameters when invoking a DCL procedure either interactively or within another DCL procedure.

Only eight(8) parameters can be passed from the command line. These parameters can contain lists of items. We'll discuss that further in the Intermediate DCL Programming session.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Parameters, Cont'd

$ @procedure_name p1 p2 p3 … p8

Notes, Cont'd:

- Reference parameters via the variable names P1 through P8.
- No built-in "shift" function. If you need it, write it as a GOSUB.

Within a procedure, you reference parameters using the symbols names P1 through P8. These symbols are local to the current procedure level.

There is no built-in "SHIFT" function that can be used to exhaust the list of parameters, as there is in UN*X and DOS. If you need this functionality, write it as a GOSUB.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Logical Names

Created using ASSIGN and DEFINE.

```
$ ASSIGN DUA0:[MY_DIR] MY_DIR
$ DEFINE MY_DIR DUA0:[MY_DIR]
```

Deleted using DEASSIGN.

```
$ DEASSIGN MY_DIR
```

DECUS Symposium - Fall 1999 San Diego Slide 42

Logical names are another kind of variable. These values can be global to a process, a job (processes owned by a parent process), a UIC group, all processes on the system, or any process which has access to the logical name table in which the logical name is defined.

The ASSIGN and DEFINE statements are similar, except for the order of their arguments.

The DEASSIGN statement is used to delete logical names.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Logical Names

Specifying a logical name table:

$ ASSIGN/TABLE=table_name

$ DEFINE/TABLE=table_name

Examples:

```
$ DEFINE/TABLE=LNM$PROCESS
$ DEFINE/PROCESS
$ DEFINE/TABLE=LNM$JOB
$ DEFINE/JOB
$ DEFINE/TABLE=LNM$GROUP    ! These require
$ DEFINE/GROUP              ! GRPNAM privilege.
$ DEFINE/TABLE=LNM$SYSTEM   ! These require
$ DEFINE/SYSTEM             ! SYSNAM  privilege.
```


Logical names can be created in any logical name table to which the process has access.

The examples here show some of the convenience qualifiers available for certain logical name tables.

Notice that privileges are required to modify certain logical name tables.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Logical Names

Specifying a mode:

```
$ ASSIGN/USER
$ ASSIGN/SUPERVISOR
$ ASSIGN/EXECUTIVE     ! Requires CMEXECprivilege
$ DEFINE/USER
$ DEFINE/SUPERVISOR
$ DEFINE/EXECUTIVE     ! Requires CMEXECprivilege
```

There is no /KERNEL qualifier. Kernel mode logicals must be created by privileged programs (requires CMKRNL privilege).

Access modes of logical names specify additional levels of privilege or supercession.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Logical Names

"Rooted" Logicals -

Specifying Translation Attributes

```
$ DEFINE/TRANSLATION_ATTRIBUTES=-
    (CONCEALED)
```

Note:

/TRANSLATION_ATTRIBUTES is a positional qualifier.

Rooted logical names provide a means of specifying a root level from which additional paths can be specified.

Positional qualifiers were discussed earlier in this session.

The next slide show examples of rooted logicals.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Logical Names

"Rooted" Logicals, Cont'd -

Example:

```
$ DEFINE/TRANS=(CONC) SRC_DIR DKA0:[SRC.]
$ DIRECTORY SRC_DIR:[000000]
$ DIRECTORY SRC_DIR:[MKISOFS]

$ DEFINE SRC_AP SRC_DIR:[AP]
$ DIRECTORY SRC_AP

$ DEFINE SRC_GL SRC_DIR:[GL]
$ DIRECTORY SRC_GL
```


This slide illustrates how to DEFINE a rooted logical and how to use rooted logicals to DEFINE other logical names.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Recurring Batch Jobs

Jobs can reSUBMIT themselves - use the F$GETQUI lexical function to obtain the needed information:

```
$ vbl = F$GETQUI( "DISPLAY_JOB", -
    item,, "THIS JOB" )
```

Useful items:

QUEUE_NAME, FILE_SPECIFICATION, AFTER_TIME, others.

Here’s where we start to discuss batch jobs that can reSUBMIT themselves.

Information that the job may need, such as the queue name, the job name, the procedure name, etc. can either be hard-coded or it can be retrieved using the F$GETQUI() lexical function by specifying “THIS_JOB” as the fourth parameter.

See the DCL Dictionary or the on-line HELP for more items that can be retrieved.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Daily Batch Jobs

Daily Jobs

Get the /AFTER time:

```
$ AFTER = F$GETQUI( “DISPLAY_JOB”,-
      “AFTER_TIME”,,-
      “THIS_JOB”)
```

Add one day to it:

```
$ NEXT = F$CVTIME( “’’AFTER’+1-”,-
      “ABSOLUTE”, )
```


Here’s an illustration of one method for getting some of the information needed to allow a batch job to SUBMIT itself for the next day.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Weekly Batch Jobs

Weekly Jobs

- Get the /AFTER time:

```
$ AFTER = F$GETQUI( "DISPLAY_JOB",-
      "AFTER_TIME",,-
      "THIS_JOB")
```

- Add seven days to it:

```
$ NEXT = F$CVTIME( "''AFTER'+7-",-
      "ABSOLUTE", )
```

DECUS Symposium - Fall 1999 San Diego Slide 49

Here's an illustration of one method for getting some of the information needed to allow a batch job to SUBMIT itself for next week.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Select Tasks by Day

Get the current day name,
look for tasks for that day.

```
$ TODAY = F$CVTIME( „ “WEEKDAY” )
$ PRC_NAME := [.‘TODAY’]WEEKLY.COM
$ FSP = F$SEARCH( PRC_NAME )
$ IF FSP .NES. “” THEN -
$ @ &FSP
```

Example Path

[.SUNDAY]WEEKLY.COM

Here’s an illustration of one method for finding tasks to be performed on a specific day.

Notice that the day name is used as the name of the subdirectory where the tasks (procedures) for that day will be found.

The symbol substitution methods used in the example will be discussed in more detail in the Intermediate DCL Programming session.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Select Monthly Tasks by Day

Get the current day number,
look for tasks for that day.

```
$ TODAY = F$CVTIME( ,, "DAY" )
$ PRC_NAME := [.'TODAY']MONTHLY.COM
$ FSP = F$SEARCH( PRC_NAME )
$ IF FSP .NES. "" THEN -
$ @ &FSP
```

Example Path

[.01]MONTHLY.COM

Here’s an illustration of one method for finding tasks to be performed on a specific day of the month.

In this case, the day number is used as the name of the subdirectory where the tasks (procedures) for that day will be found. Specifically, the tasks for the first day of the month will be run.

The symbol substitution methods used in the example will be discussed in more detail in the Intermediate DCL Programming session.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Select Last Day of the Month

Get the current day number +1, see if it’s the first of the month.

```
$ TOMORROW = F$CVTIME( “+1-”,, “DAY” )
$ IF TOMORROW .EQ. 1 THEN -
$ statement
```


Here’s an illustration of one method for determining whether the current day is the last day of the month.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Select Yearly Tasks by Date

Get the current day and numbers,
look for tasks for that day.

```
$ TODAY = F$CVTIME( ,, "MONTH" ) + -
            F$CVTIME( ,, "DAY" )  ! String values!
$ PRC_NAME := [.'TODAY']YEARLY.COM
$ FSP = F$SEARCH( PRC_NAME )
$ IF FSP .NES. "" THEN -
$ @ &FSP
```

Example Paths:

[.0101]YEARLY.COM
[.0731]YEARLY.COM

Here’s an illustration of one method for finding tasks to be performed annually.

In this case, both the month number and the day number are used as the name of the subdirectory where the tasks (procedures) for that day will be found. Specifically, the tasks for the first day of the first month and the 31st day of the seventh month will be run.

The symbol substitution methods used in the example will be discussed in more detail in the Intermediate DCL Programming session.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Q / A

Speak now or forever
hold your peas.

DECUS Symposium - Fall 1999 San Diego Slide 54

Please step to the microphone to ask your question.

One question per person, please. If you have another question, please step to the end of the line and await another turn.

DJE Systems ©1999 All Rights Reserved Introduction to DCL Programming

Thank You !

Remember to fill out the evaluation forms!

…and stay tuned for
Intermediate DCL Programming!

If evaluation forms are available, please remember to fill them out and return them to the presenter.

The Intermediate DCL Programming session will follow immediately after a short break.